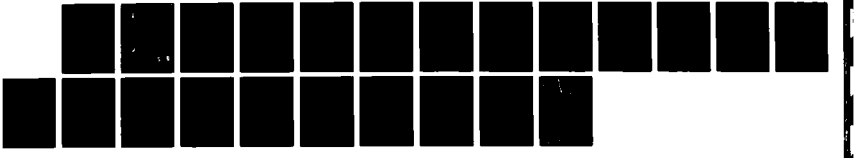
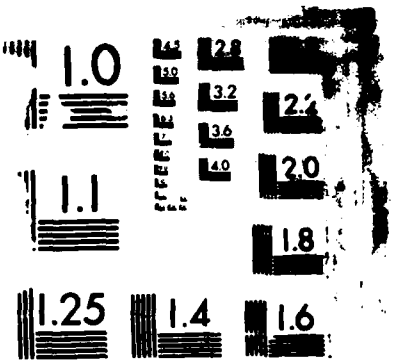
AD-A179 193 AN INTRODUCTION TO ADA (TRADE NAME) USING FORTRAN AS A BASIS(U) ARMY MISSILE COMMAND REDSTONE ARSENAL AL ADVANCED SENSORS DIR.. R K CORNWELL JAN 87 1/1

UNCLASSIFIED AMSMI/TR-RD-AS-87-2 SBI-AD-E950 964 F/G 9/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.8
3.2
3.6
4.0

TECHNICAL REPORT RD-AS-87-2

AN INTRODUCTION TO ADA USING FORTRAN AS A BASIS

Roland K. Cornwell
Advanced Sensors Directorate
Research, Development, and Engineering Center

JANUARY 1987

U.S. ARMY MISSILE COMMAND
Redstone Arsenal, Alabama 35898-5000

DISPOSITION INSTRUCTIONS

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

DISCLAIMER

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL INDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A179193

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188
Exp Date Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) RD-AS-87-2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Advanced Sensors Dir RD&E Center 6b. OFFICE SYMBOL (If applicable) AMSMI-RD-AS-RA	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Commander US Army Missile Command ATTN: AMSMI-RD-AS-RA Redstone Arsenal, Alabama 35898-5253	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
AN INTRODUCTION TO ADA USING FORTRAN AS A BASIS

12. PERSONAL AUTHOR(S)
Cornwell, Roland K.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM _____ TO OCT 86	JANUARY 1987	21

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	ADA; Programming; Computer Languages; Software; Computers.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Although almost every Electrical Engineer has heard about the new DOD programming language Ada, many still have very little knowledge of the capabilities of the language. Ada is a structured language with many new features not available in older languages. Since most engineers are familiar for FORTRAN, this report introduces Ada by making a comparison with FORTRAN. Example programs are given in both languages to illustrate some of the new features of Ada.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Roland K. Cornwell	205-876-7580	AMSMI-RD-AS-RA

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

		Page
I.	INTRODUCTION	1
II.	BACKGROUND	1
III.	OVERVIEW	3
IV.	COMPARISON OF FORTRAN AND ADA	4
	A. Sample Programs	4
	B. Statistics	9
V.	CURRENT STATUS OF ADA	13
VI.	CONCLUSION	13
REFERENCES		17

I. INTRODUCTION

This report provides engineers who have at least some working knowledge of computer programming with an introduction to the new United States Department of Defense (DOD) programming language, Ada*. Since most engineers are familiar with FORTRAN, special attention is given to comparing Ada with FORTRAN IV. Example programs are shown in both languages and the relative merits of each are discussed. FORTRAN IV was chosen over FORTRAN 77 to allow the report to be meaningful to engineers who are not familiar with the enhancements of FORTRAN 77. This should not cause any problems for FORTRAN 77 programmers since FORTRAN IV is a subset of FORTRAN 77.

While it is not designed to be a complete tutorial guide to computer programming or Ada, this report should give the reader enough information to feel comfortable when Ada is mentioned in a briefing or in a publication. Interested readers can pursue the topic by working with the examples and by studying the sources identified in the references.

The Ada examples used were developed on an IBM XT using JANUS/Ada from RR Software Inc. Since it is only a subset of Ada, JANUS/Ada is not a validated compiler. Its limitations and extensions are listed in Appendix L of the user manual [1]. A complete, validated, compiler was not available for use in writing this report.

II. BACKGROUND

As computers and microprocessors became cheaper, smaller, and more powerful, the practice of using computers in various devices became wide spread. This rapidly lead to the realization that both time and resoures must be shared between hardware and software in developing, modifying, and using such a system. In the early 1970's the DOD, and other consumers of integrated hardware and software systems, began to recognize that the expense of designing, developing and maintaining software was becoming prohibitive [2]. In many systems more resources were expended on software than hardware. While advances in technology had resulted in decreasing prices for hardware, the system costs associated with software continually increased. Figure 1 shows the forecasts resulting from studies conducted by the Electronic Industries Association (EIA) in 1980 and 1985. The graphs illustrate that as the total hardware and software market for embedded computer systems increases, the software portion of the market is increasing more rapidly than the hardware portion [3].

The flexibility of software has encouraged system designers to develop a pattern of transferring system functions from hardware to software. This allowed the same piece of hardware to be programmed to do various jobs for different customers or to perform numerous functions for the same customer. After a system was delivered, software changes and enhancements often continued indefinitely, and in some cases, for the entire life of the system. It was the ability to make these software changes that extended the useful life of many systems. This trend is expected to continue and will result in more complex, and more expensive, software systems.

*Ada is a trademark of the U.S. Department of Defense.

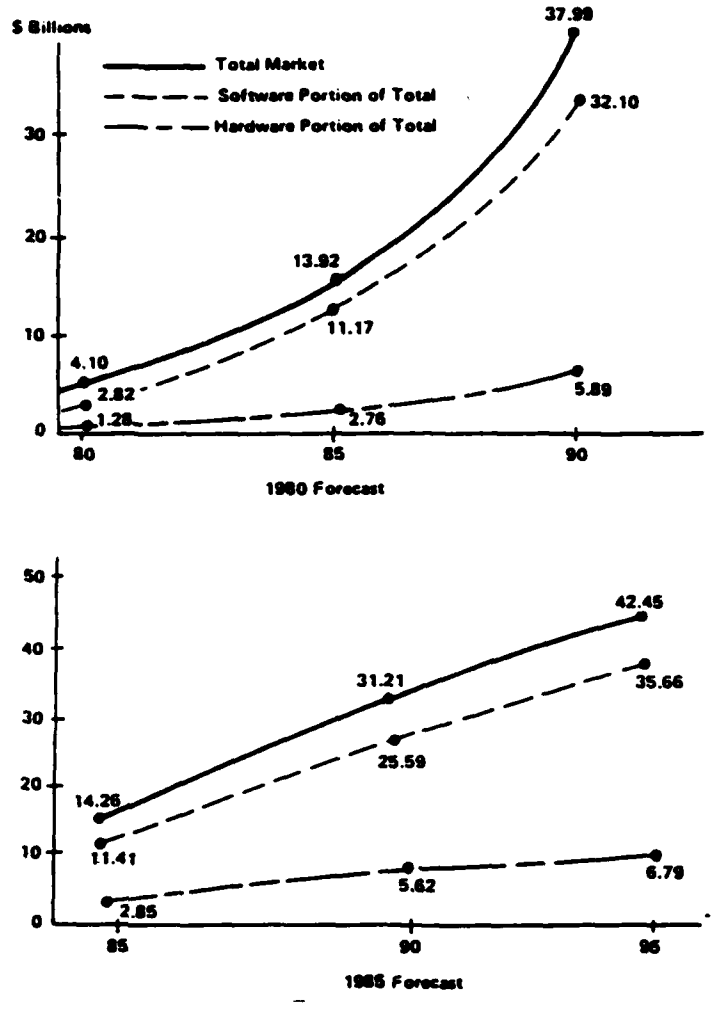


Figure 1. EIA embedded computer market forecasts [3].

The Department of Defense recognized that as a consumer of diverse systems it was spending huge amounts of money to maintain and modify the many different systems it purchased. Many times the systems that the DOD acquired arrived with software written in different assembly languages to match the various processors chosen by the systems designers. These different languages made it very difficult for users to maintain or modify the systems. Assembly language programming was bothersome enough without the added burden of remembering the idiosyncrasies of a number of different assembly languages. To solve this problem the DOD decided (in 1974) to develop a single standard language for use in its embedded computer systems.

The first attempt to set guidelines for the new standard computer programming language was published in early 1975 and called Strawman [4]. Additional studies were conducted and a version known as Woodenman was created. In June 1976, a document known as Tinman was produced that established many of the requirements for the new language. A study of various existing languages was completed to determine which would most closely match the requirements of

Tinman. Further revisions resulted in Ironman, from which seventeen contractor proposals to develop the language were generated. Four companies were chosen to proceed with their proposed designs. It is an interesting observation that each of the four companies chose Pascal to be the starting point for their efforts [5]. The final requirements document (Steelman) was produced and CII Honeywell Bull was selected to develop the language in 1979 using an international team lead by Jean Ichbiah. The new language was named Ada in memory of Augusta Ada Byron, Countess of Lovelace (1815 - 1852). She is thought to be the first programmer because of her work with Charles Babbage and his mechanical analytical engine. The ANSI standard, Reference Manual for The Ada Programming Language [6], was published in 1983 and serves as the current definition of the language [4].

III. OVERVIEW

Since Ada was patterned after Pascal, it is not surprising that much of the structure of Pascal is evident in the new language. There is a subset of Ada instructions that is decidedly similar to Pascal so it should be easy for Pascal programmers to convert to Ada [7]. Although there are some detailed differences, and some extensions in Ada, these should not present a problem.

Upon looking at an Ada program for the first time, a FORTRAN programmer may get the impression that the statements are unnecessarily long. When used properly, the longer text of Ada leads to self-documenting code and allows more error checking by the compiler. Ada was designed to be easier to read than to write since a program is written only once but may be read many times during its lifetime. A FORTRAN programmer accustomed to taking advantage of the default data types (those beginning with I, J, K, L, M, and N being INTEGER and anything else being REAL) may find the general requirement to declare the type of all variables cumbersome at first; however, this may save some errors and make programs easier to read. Some FORTRAN programmers, and some textbooks, recommend that all variables be declared even when it is not required [8]. Others rely heavily on the default data types [9,10,11].

Data abstraction may be a new concept to many users of FORTRAN. In general, abstraction may be defined to mean the process by which the functional characteristics of a facility are distinguished from the implementation of that facility [12]. This allows the programmer to define new data types such as colors, kinds of fish, or days of the week, that do not have the restrictions of working with only REAL, INTEGER, DOUBLE PRECISION, COMPLEX, LOGICAL, and CHARACTER data types. For example: A data type, BIRDS, may be declared such that there are only four legal values for variables of that type: eagles, cardinals, robins, and wrens. This would allow these birds to be referred to without resorting to mapping each species to a number.

Input and output in Ada is completely different in that there is no predefined READ statement or WRITE statement in the language. I/O must be provided by another Ada package supplied with the compiler or written by the user. In order to maintain program portability, certain standard I/O packages are required to always be available.

IV. COMPARISON OF FORTRAN AND ADA

A. Sample Programs

In order to make a comparison of FORTRAN and Ada, a sample program in each language is presented in Examples 1 and 2 respectively. The examples calculate the roots of a first or second order polynomial:

$$A*X**2 + B*X + C = 0.$$

When A is equal to zero, it will be a first order or linear equation having only one root:

$$R1 = -C / B.$$

If A is not equal to zero, the two real or complex roots of the quadratic equation will be found using [13]:

$$R1 = (-B + SQRT (B**2 - 4*A*C)) / (2*A),$$

$$R2 = (-B - SQRT (B**2 - 4*A*C)) / (2*A).$$

The FORTRAN IV program is shown in Example 1. It should be noted that the line numbers in these examples are not part of the programs but are inserted by the line editor (Edlin) used in preparing the programs. Lines 1 and 2 are simply comments to briefly describe the function of the program. Line 7 prompts the user to type the first coefficient of the equation and line 9 reads it in. Lines 11 - 16 input the other two coefficients in the same manner. Note that the use of an "*" symbol as the unit number in a WRITE or READ statement directs output to the screen and accepts input from the keyboard respectively. Line 20 checks to see if it is a linear equation. If so, lines 21 - 24 calculate and print the root of the equation and stop the program. Otherwise, lines 29 and 30 test to see if the equation has complex roots. If so, they are calculated and printed by lines 31 - 36. Otherwise, lines 42 - 45 are used to calculate and print the real roots.

Example 1. SAMPLE FORTRAN PROGRAM

```
 1: C          SAMPLE FORTRAN IV PROGRAM TO FIND THE ROOTS OF A FIRST
 2: C          OR SECOND ORDER POLYNOMIAL  A*X**2 + B*X + C = 0
 3: C
 4: C
 5: C          INPUT COEFFICIENTS OF POLYNOMIAL
 6: C
 7:       WRITE(*,1000)
 8:  1000 FORMAT(' INPUT A ')
 9:       READ(*,1003) A
10:  1003 FORMAT(F10.4)
11:       WRITE(*,1001)
12:  1001 FORMAT(' INPUT B ')
13:       READ(*,1003) B
14:       WRITE(*,1002)
15:  1002 FORMAT(' INPUT C ')
```

Example 1. SAMPLE FORTRAN PROGRAM (Continued)

```
16:       READ(*,1003) C
17: C
18: C          CHECK FOR A LINEAR EQUATION
19: C
20:       IF (A .NE. 0.0) GO TO 100
21:       X= -C/B
22:       WRITE(*,1004) X
23:  1004 FORMAT(' THE ROOT OF THE LINEAR EQUATION IS: X= ',F9.4)
24:       STOP
25:   100 CONTINUE
26: C
27: C          CHECK FOR COMPLEX ROOTS
28: C
29:       RADICL= B**2 - 4.0*A*C
30:       IF (RADICL .GE. 0.0) GO TO 200
31:       XR= -B / (2.0*A)
32:       XI= ABS( SQRT(-RADICL) / (2.0*A) )
33:       WRITE(*,1005) XR,XI
34:       WRITE(*,1006) XR,XI
35:  1005 FORMAT(' THE COMPLEX ROOTS ARE X1 = ',F9.4,' + j',F9.4)
36:  1006 FORMAT('                     AND X2 = ',F9.4,' - j',F9.4)
37:       STOP
38:   200 CONTINUE
39: C

40: C          CALCULATE REAL ROOTS
41: C
42:       X1=(-B + SQRT(RADICL)) / (2.0*A)
43:       X2=(-B - SQRT(RADICL)) / (2.0*A)
44:       WRITE(*,1007) X1,X2
45:  1007 FORMAT(' THE ROOTS ARE X1 = ',F9.4,' AND X2 = ',F9.4)
46:       STOP
47:       END
```

Program Output:

```
C>POLYNOM
INPUT A
0.0
INPUT B
3.0
INPUT C
6.0
THE ROOT OF THE LINEAR EQUATION IS: X=   -2.0000
Stop - Program terminated.
```

Example 1. SAMPLE FORTRAN PROGRAM (Continued)

```
C>POLYNOM
INPUT A
1.Ø
INPUT B
-2.Ø
INPUT C
-8.Ø
THE ROOTS ARE X1 =     4.ØØØØ AND X2 =     -2.ØØØØ
Stop - Program terminated.

C>POLYNOM
INPUT A
-1.Ø
INPUT B
-2.Ø
INPUT C
-3.Ø
THE COMPLEX ROOTS ARE X1 =    -1.ØØØØ + j    1.4142
                  AND X2 =    -1.ØØØØ - j    1.4142
Stop - Program terminated.
```

Some parts of the Ada program shown in Example 2 are similar to the FORTRAN example while other parts are very different. The "with" statement in Line 1 specifies library routines that will be needed. Notice that both upper and lower case characters are used. In general, the example Ada programs in this report will use lower case for the 63 reserved identifiers ("with", "if", "then", "loop", "for", etc.) and upper case for identifiers representing variables chosen, by the programmer. Identifiers representing packages, procedures, and functions will have only the first letter capitalized. Identifiers are analogous to FORTRAN keywords (DO, GO TO, IF, SUBROUTINE, etc.), variable names, and subroutine names. Identifiers may also be used as labels when enclosed in double angled brackets such as: <<LABEL>>. Ada is not case sensitive except when the characters themselves are under consideration such as in manipulating character strings. The programmer may choose to use upper and lower case as desired in order to make programs more readable. In general, each statement is terminated with a semicolon. A few exceptions are after: "if ... then", "else", "case ... is", and "procedure ... is", all of which have more information to follow and are terminated with a semicolon at the end of the section. For example, a semicolon is used after "end if".

Line 2 states that the name of the "package body" to follow is Polynomi. If desired, the package body may be preceded by a package specification which is not needed for this example program. The package specification would begin with: "package Polynomi is", followed by a listing of all procedures (or subprograms) and functions contained within the package. The package specification provides the interface to other programs and the package body contains the hidden details of how the package works.

A brief description of the program is given in lines 4 and 5. Comments are preceded by two dashes and may occur at the beginning of a line or after the semicolon terminating another program line.

Ada requires the type (integer, real, etc.) of each variable used in a program to be declared. Line 8 declares that the variables listed before the colon are of type float. Identifiers, including variable names, must begin with a letter, may include isolated underline characters for clarity, and are not restricted to a particular length. For example: long_sample_identifier and Long_Sample_Identifier are both legal identifiers and are considered to be the same since Ada is not case sensitive.

Line 10 gives the program immediate access to the routines, such as Sqrt, contained within the library package: Mathlib. If this "use" statement were not in the program, Sqrt could only be accessed indirectly by writing: Mathlib.Sqrt. This would indicate to the system the routine Sqrt could be found in Mathlib. Line 12 identifies the beginning of the executable portion of the program. All of the preceding statements were declarations.

Line 16 is a command to print the character string, "INPUT A", to the screen. Having read this prompt, the user is to type the value of the first coefficient on the keyboard and press return. Line 17 places the value into the variable A. Floatio.Get(A) indicates the routine Get can be found in the library package Floatio. Lines 18 - 21 input the other two coefficients in the same fashion.

Line 25 checks for a linear equation by testing for A equal to zero. If true, the root is calculated and printed by Lines 26 - 28. Notice that line 28 has more than one command on a single line. The second command, New_line, prints a carriage return and line feed so that the next output will not occur on the same line. If A is not equal to zero, control is passed to line 30 where the "else" portion of the "if" block begins. Line 34 calculates the value of B**2 - 4.0*A*C and places it in the variable RADICAL. The symbol " := ", which may be read as "becomes", takes the place of the equals symbol that is used in that position in a FORTRAN program. Line 35 determines whether the equation has complex roots. If so, lines 36 - 43 calculate and print the roots. Line 36 calculates the real part and line 37 calculates the imaginary part. Lines 38 - 40 print the first root and lines 41 - 43 print the second root. No carriage return and line feed are printed except when the New_line command is encountered at the end of lines 40 and 43. If the equation has real roots, they are calculated and printed by lines 48 - 52.

The "end if" statements on lines 53 and 54 indicate the end of the preceding "if" blocks. Notice that the "if...then" on line 35, the "else" on line 44, and the "end if" on line 53 are vertically aligned. This is a good practice as it allows the structure of the program to be clearly visible.

Finally, line 55 indicates that this is the end of the package. The identifier, Polynomi, is optional after the end, but it is a good practice to include the name here since it clarifies what is being ended. This is especially true when a package contains a number of procedures with an end statement for each.

Example 2. SAMPLE ADA PROGRAM

```
 1: with Floatio,Mathlib;
 2: package body Polynomi is
 3:
 4:      --SAMPLE ADA PROGRAM TO FIND THE ROOTS OF A FIRST
 5:      --OR SECOND ORDER POLYNOMIAL A*X**2 + B*X + C = 0
 6:
 7:
 8:      A,B,C,ROOT_1,ROOT_2,RADICAL,REAL_PART,IM_PART:FLOAT;
 9:
10:      use Mathlib;
11:
12: begin
13:
14:      --INPUT COEFFICIENTS OF POLYNOMIAL
15:
16:      Put("INPUT A : ");
17:      Floatio.Get(A);
18:      Put("INPUT B : ");
19:      Floatio.Get(B);
20:      Put("INPUT C : ");
21:      Floatio.Get(C);
22:
23:      --CHECK FOR A LINEAR EQUATION
24:
25:      if A=0.0 then
26:              ROOT_1:= -C/B;
27:              Put("THE ROOT OF THE LINEAR EQUATION IS: ");
28:              Floatio.Put(ROOT_1); New_line;
29:
30:      else
31:
32:      --CHECK FOR COMPLEX ROOTS
33:
34:              RADICAL:= B*B - 4.0*A*C;
35:              if RADICAL < 0.0 then
36:                      REAL_PART:= -B / (2.0*A);
37:                      IM_PART:= Abs( Sqrt(-RADICAL) / (2.0*A) );
38:                      Put("THE COMPLEX ROOTS ARE X1 = ");
39:                      Floatio.Put(REAL_PART);Put(" + j");
40:                      Floatio.Put(IM_PART); New_line;
41:                      Put("                    AND X2 = ");
42:                      Floatio.Put(REAL_PART);Put(" - j");
43:                      Floatio.Put(IM_PART);New_line;
44:              else
45:
46:                      --CALCULATE REAL ROOTS
47:
48:                      ROOT_1:= (-B + Sqrt(RADICAL)) / (2.0*A);
49:                      ROOT_2:= (-B - Sqrt(RADICAL)) / (2.0*A);
50:                      Put("THE ROOTS ARE : ");
51:                      Floatio.Put(ROOT_1);Put(" AND ");
52:                      Floatio.Put(ROOT_2); New_line;
53:              end if;
```

Example 2. SAMPLE ADA PROGRAM (Continued)

```
54:      end if;
55: end Polynomi;
```

Program Output:

```
C>POLYNOMI
INPUT A : 0.0
INPUT B : 3.0
INPUT C : 6.0
THE ROOT OF THE LINEAR EQUATION IS: -2.00000E+0

C>POLYNOMI
INPUT A : 1.0
INPUT B : -2.0
INPUT C : -8.0
THE ROOTS ARE :  4.00000E+0 AND -2.00000E+0

C>POLYNOMI
INPUT A : -1.0
INPUT B : -2.0
INPUT C : -3.0
THE COMPLEX ROOTS ARE X1 = -1.00000E+0 + j 1.41421E+0
                  AND X2 = -1.00000E+0 - j 1.41421E+0
```

B. Statistics

In order to compare execution times of the FORTRAN and Ada programs, a loop was inserted around the lines that compute real and imaginary parts of an equation that has complex roots. These changes can be seen at line numbers 31 - 37 of Example 3 and line numbers 37 - 42 of Example 4. This causes those lines to be executed 30,000 times before the results are printed. The amount of time required to compile and link these programs using a batch file, and the execution times, are shown in Table 1.

Example 3. FORTRAN PROGRAM WITH LOOP FOR RUN TIME COMPARISONS

```
 1: C          SAMPLE FORTRAN IV PROGRAM TO FIND THE ROOTS OF A FIRST
 2: C          OR SECOND ORDER POLYNOMIAL  A*X**2 + B*X + C = 0
 3: C
 4: C
 5: C          INPUT COEFFICIENTS OF POLYNOMIAL
 6: C
 7:       WRITE(*,1000)
 8:  1000 FORMAT(' INPUT A ')
 9:       READ(*,1003) A
10:  1003 FORMAT(F10.4)
11:       WRITE(*,1001)
12:  1001 FORMAT(' INPUT B ')
13:       READ(*,1003) B
```

Example 3. FORTRAN PROGRAM WITH LOOP FOR RUN TIME COMPARISONS (Continued)

```
14:         WRITE(*,1002)
15:   1002 FORMAT(' INPUT C ')
16:         READ(*,1003) C
17: C
18: C           CHECK FOR A LINEAR EQUATION
19: C
20:         IF (A .NE. 0.0) GO TO 100
21:         X= -C/B
22:         WRITE(*,1004) X
23:   1004 FORMAT(' THE ROOT OF THE LINEAR EQUATION IS: X= ',F9.4)
24:         STOP
25:    100 CONTINUE
26: C
27: C           CHECK FOR COMPLEX ROOTS
28: C
29:         RADICL= B**2 - 4.0*A*C
30:         IF (RADICL .GE. 0.0) GO TO 200
31: C
32: C           LOOP INSERTED TO COMPARE EXECUTION TIMES
33: C
34:         DO 300 I=1,30000
35:         XR= -B / (2.0*A)
36:         XI= ABS( SQRT(-RADICL) / (2.0*A) )
37:    300 CONTINUE
38: C
39:         WRITE(*,1005) XR,XI
40:         WRITE(*,1006) XR,XI
41:   1005 FORMAT(' THE COMPLEX ROOTS ARE X1 = ',F9.4,' + j',F9.4)
42:   1006 FORMAT('                     AND X2 = ',F9.4,' - j',F9.4)
43:         STOP
44:    200 CONTINUE
45: C
46: C           CALCULATE REAL ROOTS
47: C
48:         X1=(-B + SQRT(RADICL)) / (2.0*A)
49:         X2=(-B - SQRT(RADICL)) / (2.0*A)
50:         WRITE(*,1007) X1,X2
51:   1007 FORMAT(' THE ROOTS ARE X1 = ',F9.4,' AND X2 = ',F9.4)
52:         STOP
53:         END
```

Program Output:

```
C>POLYNLOP
INPUT A
-1.0
INPUT B
-2.0
INPUT C
-3.0
THE COMPLEX ROOTS ARE X1 =    -1.0000 + j    1.4142
                  AND X2 =    -1.0000 + j    1.4142
Stop - Program terminated.
```

Example 4. ADA PROGRAM WITH LOOP FOR RUN TIME COMPARISONS

```
 1: with Floatio,Mathlib;
 2: package body Polyloop is
 3:
 4:      --SAMPLE ADA PROGRAM TO FIND THE ROOTS OF A FIRST
 5:      --OR SECOND ORDER POLYNOMIAL A*X**2 + B*X + C = 0
 6:
 7:
 8:      A,B,C,ROOT_1,ROOT_2,RADICAL,REAL_PART,IM_PART:FLOAT;
 9:
10:      use Mathlib;
11:
12: begin
13:
14:      --INPUT COEFFICIENTS OF POLYNOMIAL
15:
16:      Put("INPUT A : ");
17:      Floatio.Get(A);
18:      Put("INPUT B : ");
19:      Floatio.Get(B);
20:      Put("INPUT C : ");
21:      Floatio.Get(C);
22:
23:      --CHECK FOR A LINEAR EQUATION
24:
25:      if A=0.0 then
26:              ROOT_1:= -C/B;
27:              Put("THE ROOT OF THE LINEAR EQUATION IS: ");
28:              Floatio.Put(ROOT_1); New_line;
29:
30:      else
31:
32:      --CHECK FOR COMPLEX ROOTS
33:
34:              RADICAL:= B*B - 4.0*A*C;
35:              if RADICAL < 0.0 then
36:
37:      --LOOP INSERTED TO COMPARE EXECUTION TIMES
38:
39:                              for I in 1..30000 loop
40:                      REAL_PART:= -B / (2.0*A);
41:                      IM_PART:= Abs( Sqrt(-RADICAL) / (2.0*A) );
42:                              end loop;
43:
44:                      Put("THE COMPLEX ROOTS ARE X1 = ");
45:                      Floatio.Put(REAL_PART);Put(" + j");
46:                      Floatio.Put(IM_PART); New_line;
47:                      Put("                      AND X2 = ");
48:                      Floatio.Put(REAL_PART);Put(" - j");
49:                      Floatio.Put(IM_PART);New_line;
50:              else
```

Example 4. ADA PROGRAM WITH LOOP FOR RUN TIME COMPARISONS (Continued)

```
51:
52:                         --CALCULATE REAL ROOTS
53:
54:                         ROOT_1:= (-B + Sqrt(RADICAL)) / (2.0*A);
55:                         ROOT_2:= (-B - Sqrt(RADICAL)) / (2.0*A);
56:                         Put("THE ROOTS ARE : ");
57:                         Floatio.Put(ROOT_1);Put(" AND ");
58:                         Floatio.Put(ROOT_2); New_line;
59:                 end if;
60:         end if;
61: end Polyloop;
```

Program Output:

```
C>POLYLOOP
INPUT A : -1.0
INPUT B : -2.0
INPUT C : -3.0
THE COMPLEX ROOTS ARE X1 = -1.00000E+0 + j 1.41421E+0
                  AND X2 = -1.00000E+0 - j 1.41421E+0
```

TABLE 1. Compile and Execution Times (minutes:seconds).

	FORTRAN	ADA
COMPILE AND LINK	1:32	2:56
EXECUTION	0:22	0:37

The size of the disk files occupied by these programs is shown in Table 2. It should be understood that the comparison of these two example programs does not represent a statistical sample and general conclusions may not be drawn from this data. Also, it is well known that when new languages are designed the first compilers written for them may not be very efficient. After many years of experience in writing compilers for a new language have been acquired, better compilers will usually become available. For example, the FORTRAN compilers available today are much better than the first FORTRAN compilers that were developed. General conclusions about the efficiency of the two languages should be made only after testing a number of different programs compiled and run on various machines.

TABLE 2. Size of Disk Files (bytes).

	FORTRAN	ADA
SOURCE FILE	1331	1317
EXECUTABLE FILE	40122	22272

V. CURRENT STATUS OF ADA

The DOD is very concerned with enforcing the Ada standard and has copyrighted the name "Ada" in order to maintain control of the language. To assure standardization, a compiler must pass more than 2500 validation tests before it is allowed to use Ada as its title. Revalidation must be completed annually at a cost currently estimated to be $75,000 per compiler [14]. Validation of new compilers and revalidation of old compilers is currently the responsibility of the Ada Joint Project Office (AJPO). Some changes in the validation requirements are being considered as the number of compilers in need of validation increases [15].

Ada was standardized as ANSI/MIL-STD-1815A in February 1983. This was eight years after DOD first began to define the technical requirements for the new language. At that time it was difficult for people to begin to learn the language due to the lack of available compilers. Although some critics thought it could not be done, two compilers had been validated by the end of 1983: one developed by Data General under license from Rolm Corp., and one from Western Digital Corp. [14]. By mid-1984 only about a half-dozen Ada compilers had been validated. In November 1985 the number had grown to 16. By April 2, 1986 there were 29 validated Ada compilers available [3]. A list of these 29 validated compilers was put together by Defense Electronics in July 1986 and is shown in Figure 2.

VI. CONCLUSION

Programmers who are only familiar with FORTRAN have fallen behind the current software technology and would probably benefit from putting forth the effort required to learn a new structured language such as Ada. Users of newer languages, such as ALGOL and especially PASCAL, will find Ada easier to use. However, there are enough similarities between FORTRAN and Ada to allow FORTRAN programmers to understand Ada with a reasonable effort. When beginning to use Ada, FORTRAN programmers must exercise care not to ignore all of the new features available in the language and simply create FORTRAN programs within the syntax of Ada.

As can be seen from the previous example programs, Ada is a general purpose language which may be used for standard data processing needs in spite of the fact that it was originally designed for use in embedded computer applications. The new features of Ada such as data abstraction, tasking, strong data typing, exception handling, and its readability, are expected to encourage its use outside of its original target area. FORTRAN certainly has not lost its usefulness since it has been so widely used and there is an enormous library of scientific algorithms available in the language. However, Ada can increase a FORTRAN programmers capabilities and open up a complete new world of software concepts.

Vendor & Compiler	Host Machine & Operating System	Target Machine & Operating System	Test Suite Version	Date Validated
Alsys (France) AlsyCOMP001 Version 1.3	VAX 11/750 (VMS 4.1)	ALTOS ACS 68000 14 (ALTOS Version 1)	1.6	11-08-85
Alsys (France) AlsyCOMP002 Version 1.0	HP 9000 200/220 & 300/320 (HP-UX V5.0)	Same as Host	1.6	11-02-85
Alsys (France) AlsyCOMP004 Version 1.0	Apollo DOMAIN DN460, DN360, DSP80A (AEGIS Vers. SR9)	Same as Host	1.6	11-03-85
Alsys (France) AlsyCOMP005 Version 1.0	SUN Workstations 2/120, 2/50, 3/160 (SUN UNIX 4.2)	Same as Host	1.6	11-02-85
Alsys (France) AlsyCOMP006 Version 1.0	VAX 11/750 (VMS 4.1)	IBM PC-AT (MS-DOS 3.1)	1.6	12-05-85
Air Force Armament Lab. AFATL Ada Cross Compiler 1.0	CDC Cyber 170/760 (NOS 2.4)	Zilog Z8002 Development Module (Dev. Module Monitor Program)	1.6	10-15-85
Data General Corp. ADE Ada Compiler	DS/4000, DS/4200, MV/4000-DC, MV/4000, MV/8000-C, MV/8000-II, MV/10000, MV/10000SX, all using AOS/VS 5.04	All host architectures plus the following machines using AOS/RT32 4.01: MV/4000, MV/8000-II MV/10000, MV/10000SX.	1.5	05-18-85
DDC International DDC Ada Compiler System	VAX 11/785 (VMS 4.1)	Same as Host	1.6	11-26-85
Department of the Army ALS AdaVAX Version 2.47 (same as SofTech)	VAX 8600, VAX11/780, VAX 11/785 (VMS 4.1) MicroVAX II (MicroVMS 4.1M)	All Host Configurations	1.6	10-17-85
Digital Equipment Corp. DEC VAX-Ada Compiler V.1.1	VAX 8600, VAX 11/785, VAX 11/782, VAX 11/780, VAX 11/750, VAX 11/730 (VMS 4.2) MicroVAX I&II, VAX station I&II (MicroVMS 4.2) MicroVAX II (VAXELN 2.0)	All Host Configurations	1.6	09-06-85
Honeywell Information Systems GCOS6 Ada Compiler Version 1.1	DPS 6/95, DPS 6/94, DPS 6/75, DPS 6/74 & DPS 6/70 (MOD 400, Release 3.0 & 3.1) DPS 6/85 (MOD 400 Release 3.1)	All Self-targeted Hosts DPS 6/75 (Cross-Compiler from DPS 6/95 & DPS 6/85 Host	1.6	11-29-85
Honeywell Large Systems GCOS-8 Version 3.1	DPS-88 (SR2300 (7/85 IFAD B.4 SMAS BO)	Same as Host	1.6	12-20-85
Intermetrics Inc. I2Ada Compiler, Version 17.08	IBM 370 architecture (IBM 3083, Model BX2 or IBM 4341, Model L2) (UTS 2.3)	Same as Host	1.6	12-10-85
Rational Machine, Inc.	R1000 (Rational Environment, A.2.0.6)	Same as Host	1.5	05-24-85
Rolm Corp. ADE Ada Version 2.30.03.12	Rolm MSE/800 AOS/VS 4.04	Same as Host	1.5	05-24-85
SofTech AdaVAX Version 2.47 (same as Dept. of the Army)	VAX 8600, VAX 11/780, VAX 11/785 (VMS 4.1) MicroVAX II (MicroVMS 4.1M)	All Host Configurations	1.6	10-17-85
SofTech, Inc. Ada86 Version 1.21	VAX 11/780, VAX 11/785 (VMS 4.1)	INTEL 8086 on 86/30 board & INTEL 80186 on 186/03A board	1.6	11-14-85

Figure 2. Validated Ada compilers on April 2, 1986 [3]. (sheet 1 of 2)

Vendor & Compiler	Host Machine & Operating System	Target Machine & Operating System	Test Suite Version	Date Validated
Systeam/German MOD, VAX-11	DEC VAX 11/750 (VMS 4.1)	Same as Host	1.6	11-24-85
*TeleSoft Inc. TeleSoftAda Version 2.0a6	VAX 11/780 (UNIX 4.2 BSD) & (VMS 3.4)	Same as Host	1.4	02-05-85
TeleSoft Inc. TeleSoft-Ada Version 2.3C3	Gould Power Node Model 9750 (Gould UTX, V1.1)	Same as Host	1.6	12-06-85
TeleSoft Inc. TeleSoft-Ada Version 2.3C3	Gould Concept/32 Model 6750 (Gould MPX, V3.2)	Same as Host	1.6	12-06-85
TeleSoft Ada Compiler Version 2.39	Gould PS3000 Workstation (CSD UTX/3000)	Gould Concept/32 Model 6750 (MPX 3.2)	1.6	11-08-85
TeleSoft Ada Compiler Version 2.0a6	Gould Power Node Model 9050 (Gould UTX, V1.1)	Same as Host	1.6	12-05-85
Verdix Ada Compiler VAda-010-0101 Version V03.04	VAX 11/750 (UNIX 4.2 BSD)	Same as Host	1.5	03-15-85
Verdix Corp. VADS, VAda-010-1010 Version V05.00	SUN Microsystems Model 2/120, (Berkeley UNIX 4.2 BSD, Release 1.1)	Same as Host	1.5	06-07-85
Verdix Corp. VADS Version V03.06	VAX 11/785 (ULTRIX 1.0)	Same as Host	1.5	06-14-85
Verdix Corp. VADS Version V5.2	VAX 11/750 (VMS 4.1)	Same as Host	1.6	11-17-85
Verdix Corp. VADS Version V5.2	Tektronix 6130 (UTek, Release 2.1.1)	Same as Host	1.6	11-16-85
Verdix Corp. VADS Version V5.2	Sequent Balance (Sequent DYNIX, Release 1.3.2)	Same as Host	1.6	11-15-85
Verdix Corp. VADS Version V5.2	CCI Power 6/32 (Power 6 UNIX, Release 1-11)	Same as Host	1.6	11-16-85

*Re-validation pending.

Figure 2. Validated Ada compilers on April 2, 1986 [3]. (sheet 2 of 2)

REFERENCES

1. JANUS/Ada PACKAGE USER MANUALS, RR Software, Inc. Madison, Wisconsin, 1983.

2. Carlson, William E., Ada: A Promising Beginning, Computer, June 1981.

3. Taking a Hard Line on Software, Defense Electronics, July 1986.

4. Barnes, J. G. P., Programming in Ada, Addison-Wesley Publishing Co., 1984.

5. Gehani, Narain, Ada An Advanced Introduction, Prentice-Hall, Inc., Englewood Ciffs, N.J., 1983.

6. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983,

7. Pyle, I. C., The Ada Programming Language, Prentice-Hall International Inc., 1 981.

8. Page, Rex and Didday, Rich; FORTRAN 77 for Humans, West Publishing Co., St. Paul, Minnesota, 1980.

9. Schwar, James P. and Best, Charles L., Applied FORTRAN for Engineering and Science, Science Research Associates, Inc., 1982.

10. McCracken, Daniel D., A Guide To FORTRAN IV Programming, second edition, John Wiley & Sons, Inc., 1972.

11. Lipschutz, Seymour and Poe, Arthur, Theory and Problems of Programming with FORTRAN, Schaum's Outline Series, McGraw-Hill Book Co., 1978.

12. Brender, Ronald F. and Nassi, Isaac R., What is Ada?, Computer, June 1981.

13. Spiegel, Murray R., Advanced Mathematics for Engineers and Scientists, Schaum's Outline Series, McGraw-Hill Book Co., 1971.

14. Mosley, J. D., Validated Compilers, Software Tools Ease Ada Program-Development Tasks, EDN, August 22, 1985.

15. Suydam, William E., Jr., Proliferation of Ada Compilers Keeps Validation Services Running, Computer Design, February 1, 1986.

DISTRIBUTION

	Copies
US Army Materiel System Analysis Activity ATTN: AMXSY-MP Aberdeen Proving Ground, MD 21005	1
ITT Research Institute ATTN: GACIAC 10 W. 35th Street Chicago, IL 60616	1
AMSMI-RD, Dr. McCorkle	1
Dr. Rhoades	1
-RD-AS, Mr. Powell	1
Mr. Todd	1
Mr. Pittman	1
-RD-AS-RA, Dr. Loomis	1
Mr. Cornwell	25
-RD-CS-R	15
-RD-CS-T	1
AMSMI-GC-IP, Mr. Bush	1

END

5-87

DTIC